Improving the Performance of Hierarchical Scheduling for Rendering

Ab Rouf Khan[1], Mir Saleem[2], Shoaib Amin Banday[3]

[1] M Tech, School of Computing Sciences and Engineering. VIT University, Vellore - 632014 Tamil Nadu.
[2]Assistant Professor, Department of computer sciences, Amar Singh College, Srinagar.
[3]Member IEEE, Department of Electronics and Communication Engineering, NIT Srinagar, J&K India-190006
khanrouf25@gmail.com,arsaleemmir@gmail.com,shoaibee.a@gmail.com

Abstract **– With the improvement and development of the high performance computing systems, the need arises to use the resources available at hand efficiently. Rendering is one kind of application which is suitable for high computing. In the modern era of computing we have got the quad processors available, but the processes and data we manipulate on these, is still based on the serial algorithms in many of the cases. As rendering needs more computing and is associated with huge data access, it can be broken into the smaller subtasks of same nature to be executed on the different processors. The parallel approach works on the principle of solving or computing the different similar sub-tasks using the different available resources of computing (processors) in a parallel fashion. Means at the end of one unit of time, we are having the result of as much number of subtasks in hand, as the number of processors available. The only thing we need to keep in mind is how to efficiently share the resources between different subtasks and how to balance the load factor as per different processors are considered. In this paper we are proposing a hierarchical scheduling policy for rendering to improve its performance as compared to the present day available methods. We will evaluate the approach with the existing approaches for rendering to show the improvement in performance.**

Keywords **– Simplicity, Beauty, Elegance, Rendering, Parallel Processing.**

I INTRODUCTION

The paper aims at implementing the large scale rendering using the hierarchical approach in a parallel paradigm. Our basic aim is to improve the performance of the current available methods of rendering under the hierarchical domain.

A. Motivation

Rendering being a fundamental and basic aspect of Computer Graphics, the improvement in the performance of this basic process can prove to be very fruitful for the field of graphics. Also considering the fact, that in current age of technology, we have the computers available with the facility of more than one processors, the efficient utilization of the same is very important. [2] The resources available at hand in the modern era will be used efficiently only once we start writing the parallel programs to be executed on these parallel machines. The hierarchical methods available currently can be converted into the more efficient algorithms, once we modify these for the parallel scenario. [4] What motivated the authors the most is the fact that the hierarchical method of rendering is itself efficient in nature; further improvements will surely lead to a better performance.

B. Objective

The performance of the large scale rendering using hierarchical approach can be improved by implementing the same in the parallel environment. The task queue which contains the different tasks to be carried out in the process of hierarchical rendering, can be divided into a set of independent tasks.

Each set of independent tasks is to be assigned to each core of the „multi-core machine available. The tasks will be carried out simultaneously by the different processors. One processor will act as a controlling processor, which will pass the computations (if required) from one processor to the other and monitor the overall system. The procedure will be more efficient once we reduce the computation overhead, which in turn will depend upon the division of the task set from the task queue. As in case of rendering, there are many independent sub processes to be carried out, the task set can be easily divided into a set of independent tasks.

II. RELATED WORK

The paper builds around the research work carried out in the fields of the Computer Graphics by many authors. The related work carried out prior to this paper includes the hierarchical rendering, parallel rendering, and basic rendering procedures. Luyang Dong, Bin Gong and Yan Ma, Yi Hu [1] in the year 2011 came up with a work related to the Hierarchical scheduling in the paper titled as "A Hierarchical Scheduling Policy for Large-scale Rendering" [1]. In the paper the authors have given an algorithm for Hierarchical scheduling and the load balancing between different nodes based on the feedback of the runtime information. They divided the whole policy of rendering broadly into three vertical layers which included task selection, resource dispatching and load balancing layers [1]. Also in year 2005 the developers at HP labs - Yunhong Zhou, Terence Kelly, Janet Wiener, and Eric Anderson in the paper titled "An Extended Evaluation of Two-Phase Scheduling Methods for Animation Rendering" produced the work on scheduling animation rendering jobs. They developed an efficient 2-phase scheduling algorithm for animation rendering and proved it both theoretically as well as using the simulators that this 2-phase scheduling algorithm is better than the previous available scheduling algorithms for rendering. [10] Apart from scheduling, the Load distribution is also an important parameter used in large scale rendering. To address this issue of Load balancing, the professors of Concordia University, Canada – Alexandre Beaudoin, Dhrubajyoti Goswami,and Sudhir Mudur came up with a work titled in the paper "Two-phase Load Distribution for Rendering Large 3D Models on a Graphics Cluster". The paper aimed at addressing the problem of distributing the computations based in rendering for a real-time display of very large 3D models. They used a programmable graphics processing unit at each node and divided the rendering computations in two phases using two separate GPU programs. [3] Another aspect of the rendering is the parallel rendering. In this regard in year 2006, Ping Yin, Xiaohong Jiang, Jiaoying Shi and Ran Zhou came up with a work in the paper titled "Multi-screen Tiled Displayed, Parallel Rendering System for a Large Terrain Dataset". In this paper the authors proposed a parallel rendering system for a large terrain dataset. They used the technique of „multi-screen tiled display to show the rendering result they obtained after implementing a parallel approach of rendering on a large data set.[5] They came up with the implementation of famous Level of Core (LOD) algorithms of 1990s. They also used Graphics optimization LOD algorithms to improve the displaying of the rendering result obtained [5].

III. SYSTEM DESIGN

The system proposed is designed to perform the task of large scale rendering in a parallel environment. The system consists of a task queue in which the tasks are divided into independent set of tasks. Another important component of the proposed system is the multi core architecture which consists of a number of processors with each processor having its own memory. The output from the different cores after processing is fed to the workstations. Workstations consist of the different resources internally like Graphics Card etc. The different resources present in the workstations can be used by the independent rendering processes in parallel. Also the inclusion of more workstations makes the system more generic. The resource to a particular task can be allocated from any of the workstation resources available. This will greatly reduce the load balancing problem. The load balancing problem is solved efficiently with respect to the each node. Figure 1 shows the architecture of the proposed system.

Rendering is a very complex task which needs very large amount of computations. So, here we will divide the large complex task into „n independent task sets. These „n task sets are accommodated into single task queue. From that queue each task set is assigned to corresponding processors. Suppose, we have „p processors and n = p then each task is assigned to each processor. Otherwise if p < n then task is assigned to the processors according to n/p fashion. Here each processor consists of its memory and core. After assigning task to the processor, processor will select particular resource from the pool of workstations in parallel fashion according to the need of resource for rendering. In this way, tasks are assigned in parallel manner to each resource.

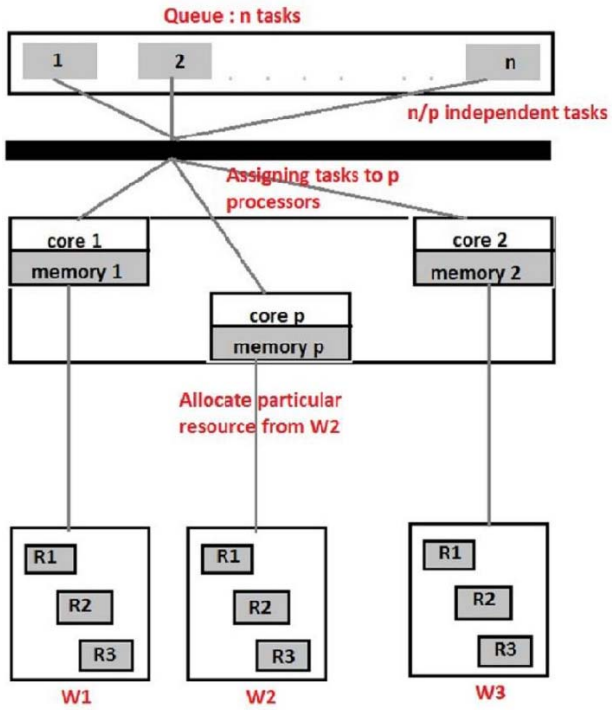

Figure 1. Proposed System Design

Serial Algorithm	Parallel Algorithm

• Q : Task Queue
• S : no tasks in queue
• P_{si} : Proportion of s to be run on node i
• R_s : Resource allocated to s
 	
• order tasks in *Q*
• for each task *s in Q*
• Perform resource selection, get *Rs*
• for each node *i in Rs*
• compute *Psi*
• end for
• put *s into operation on Rs based on resource*
• dispatching policy used.
• end for
	• Q : Task Queue
	• n : no tasks in queue
	• P : no of cores in a multicore processor.
	• $A_{r\,n/p\,w}$: Amount of independent task set to be run on workstation 'w'.
	• $R_{r\,n/p\,}$: Resource allocated to the task.
	• Order $_{r}n/p_{]}$ no of independent tasks set in Q
	• for each independent task set in Q
	  • Assign each independent task to a Core C_i
	  • for each workstation available
	  • Perform resource selection to obtain $R_{r\,n/p\,}$
	  • Complete $A_{r\,n/p\,w}$
	  • End for
	  • Put $_{r}n/p_{]}$ independent task into operation on Rw based on the dispatching policy used
	• End for

Table 1. Parallel v/s Serial Algorithm for Rendering

A. Comparision between Parallel and Serial Algorithm for Rendering

The serial and parallel algorithms for rendering are given in the Table 1. In the serial algorithm the tasks are present in the task queue Q. S is one of the tasks present in the task queue. The algorithm starts with ordering the tasks from in the task queue Q. All the tasks from the task queue Q are assigned the resources it needs after the proper resource selection. The resource selection procedure is responsible for selecting the appropriate resource which a particular task needs to complete its execution. It is worth to mention here that the resource allocation and selection is done serially. After allocating the resources to the tasks, we need to calculate how much resources are needed by a particular task with respect to a workstation. In other words we need to find out how much work is being done by each node or each workstation. After the task has utilized the resources from a particular node it needs, it is set into operation based on the dispatching policy needed. The time complexity of the algorithm depends on the number of the tasks and the number of the nodes available. For the sake of symmetry let us take the number of tasks and the number of nodes available both as „n□. Therefore the time complexity of the algorithm is n*n = n2. Using the asymptotic notation we can write the time complexity as **O (n2).** The time complexity comes out to be as O (n2) because there are two „for□ loops running in the algorithm, and the second „for□ loop is internal to the first one. The Parallel rendering algorithm acts in a different way as compared to the serial one. The first differentiation between the serial and parallel algorithm starts from the fact that the tasks in the task queue are ordered into the independent set of „task-sets□ rather than the tasks itself.

The independent task sets may contain much number of the individual tasks, but these tasks are similar to one another in some way. That means the tasks within a task set are similar to each other. However one task set is independent from of the task set. The queue of the tasks is divided into the number of task sets based on the number of the processors available in the multi-core architecture. After the independent task sets are identified, each independent task set is assigned to the processors for the pre-processing. The output from these different cores after pre-processing is fed to the different resources available at the workstations in parallel. This point needs a further explanation here. What we actually mean that there are different workstations available with different resources embedded in them. Once the independent tasks-set needs a particular resource from any workstation, the resource from that workstation will be assigned to the particular task set and the particular task set will be put into the operation with respect to the particular resource based on the dispatching policy used. The time complexity of the parallel algorithm will depend upon the number of the processors used and the business logic of the algorithm.

One important point worthy to mention here is that there is one important component which plays an important role in deciding the complexity of a parallel algorithm and that is:

„Communication Overhead□. Communication overhead is the time taken by the processors to exchange the data between the different tasks. The communication overhead increases as the dependency among the tasks increases because they need to share more data and information. However the parallel algorithm for rendering given above is designed in such way that the task-sets assigned to the different processors are independent. This means the communication overhead is almost eliminated entirely. The time complexity of the parallel algorithm will be thus much lesser than the serial one. Suppose the number of independent task sets is „m□ and the number of the workstations will remain the same as in the case of serial algorithm i.e., „n□. It is obvious that m<n always. Assume the number of processors is „p□. Then the time complexity of the parallel rendering algorithm is [(m*n)/p]. Using the asymptotic notation the time complexity can be written as: **O [(m *n)/p]**.

Since time complexity is the major deciding parameter in the algorithmic complexity analysis, and the algorithm with lesser time complexity is regarded as better than the corresponding more complexity one, the parallel rendering algorithm is better than the serial rendering algorithm. From the above discussion we conclude that the Parallel Rendering Algorithm proposed in this paper is much efficient than the Serial Rendering Algorithm. And since the better algorithm results in better throughput, the Rendering can be done efficiently using the above proposed Parallel Algorithm.

Speed-up: The speed-up of Serial Rendering Algorithm v/s Parallel Rendering Algorithm is defined as: Serial Execution Time / Parallel Execution Time.

$$\text{Speed-up} = \frac{Serial\ Execution\ Time}{Parallel\ Execution\ Time}$$

$$= \frac{\theta(n2)}{\theta[(m*n/p)]}$$

$$= p*(n/m)$$

Where: n is no. of tasks used in task queue in case of Serial Rendering Algorithm m is the number of Independent task sets in case of Parallel Rendering Algorithm m < n

Efficiency: Efficiency is calculated on the basis of formula given below:

$$\text{Efficiency} = \frac{Speed-up}{Number\ of\ processors}$$

$$= \frac{P(n/m)}{P}$$

$$= n/m$$

Since the value of „m□ is always going to be lesser than the value of „n□, the efficiency is better than the existing serial Rendering algorithm.

IV. CONCLUSION

Based In this paper a Parallel Rendering algorithm to improve the performance of Hierarchical large-scale rendering is put forward. The load balancing between task selection and resource dispatching algorithms are imported from the previous works done by many authors. The parallel Rendering Algorithm is compared v/s the corresponding Serial Rendering Algorithm and the considerable improvements are found to exist. Based on the calculation of the important parameters like Speed-up and Efficiency it can be concluded that the parallel algorithm is much efficient in terms of both time complexity and speed-up.

V. REFERENCES

1. Luyang Dong,Bin Gong, Yan Ma, Yi Hu, A Hierarchical Scheduling Policy for Large-scale Rendering, Sixth Annual China Grid Conference, 2011.
2. Hansen, C.; Krogh, M.; Painter, J., Parallel rendering techniques for massively parallel visualization, IEEE conference, 1997.
3. Yi Zhu. , A Method for Large-Scale Terrain Rendering Based-on GPU, 2nd International Conference on multimedia and information Technology, 2010.
4. Sakamoto, N.; Nonaka, J.; Koyamada, K.; Tanaka, S., Particle-Based volume rendering, IEEE Conference, 2007.
5. Ping Yin, Xiaohong Jiang, Jiaoying Shi and Ran Zhou, Multi-screen Tiled Displayed, Parallel Rendering System for a Large Terrain Dataset, The International journal of Virtual Reality, 2006.
6. Yunhong Zhou, Terence Kelly, Janet Wiener, and Eric Anderson, An Extended Evaluation of Two-Phase Scheduling methods for Animation Rendering, Processing of the 2005 workshop on job scheduling Strategies for parallel Processing (JSSPP□05), 19 June 2005.